Updated Methods for US Census API Data Extraction in R

Scott Chase, Faculty Mentor: Professor Zack Almquist

Summer 2015

Introduction

Beginning in early 2014, I started work with Professor Zack Almquist in the University of Minnesota Twin Cities Department of Sociology and School of Statistics to develop a convenient way to access the wealth of demographic data made available by the United States Census Bureau in the statistical computing language R. The primary method for extracting this data in both versions of CensusAPI2000 and CensusAPI2010 from the Census Bureau is the Application Programming Interface (API), a method for systematically extracting specific data from an organization's database. After developing the first versions of the functions CensusAPI2010 and CensusAPI2000, several problems were encountered. The first generation of CensusAPI2010 and CensusAPI2000 accessed far more information than necessary to return what the user requested through accessing various R packages, resulting in large inefficiencies. Secondly, far too many API calls were made, often resulting in unnecessary errors from the Census Bureau's API and long wait times especially at smaller geographic levels. Lastly, the previous version of CensusAPI2010 was only able to gather data for one state at a time, while the new version can gather data for multiple states in one function call, returning one data frame with all of the requested data for each state. A new generation of the functions CensusAPI2000 and CensusAPI2010 that is more efficient, reliable, and faster has

been developed. While this new generation includes two functions to collect data from both the 2000 and 2010 decennial censuses, only the 2010 version will be outlined here as the two versions are quite similar.

FIPS Codes

The federal government partitions the entire United States into geographic areas, which ordered from largest to smallest are state, county, tract, block group, and block. The 50 states are broken down into over 11 million blocks, which in conjunction can paint a finely grained demographic picture of the United States. For instance, the Empire State Building in New York City sits in the geographic region identified by FIPS "360610076001001," i.e. state "36," county "061," tract "007600," block group "1," and block "1001." It is important to note that given the whole 15-digit FIPS code, it is possible to deduce the state, county, tract, block group, and block. There is an additional class of FIPS codes for CDP (Census Designated Places), which are geographic regions such as small towns that have schools, roads, police departments, etc, and are useful for analysis but aren't necessarily classified as a city or township by the state.

Improving Efficiency

In the first versions of CensusAPI2000 and CensusAPI2010, every call to the function would result in the loading of one of the R packages maintained by Professor Almquist. Depending on the geographic level, a specific R package had to be loaded, and then depending on the state, a specific set of data contained within that R package would be loaded. The FIPS codes contained in this data set would be

extracted, and demographic data from the Census API matched to them. As the user selects smaller geographic levels, the size of the R package necessary to gather the FIPS codes becomes larger, resulting in large inefficiencies. To solve this problem, FIPS codes for each state were pre-gathered and compiled into one large data set, from which every FIPS code at every geographic level in the United States can be deduced. This removes the need for loading unnecessary data and simplifies the collection of appropriate FIPS codes greatly. For example, see below the first few rows and columns of the unified FIPS data set. Through the use of the substr() function in R, it is possible to figure out which state, county, tract, block, and block group is described by each 15-digit FIPS code: the first two digits designate the state, the 3rd through the 5th digits designate the county, the 6th to the 11th digits designate the tract, the 12th digit identifies the block group, and the 12th through the 15th digits represents the block.

```
##      01                 02                 04
## [1,] "010010201001000" "020130001001000" "040019426001000"
## [2,] "010010201001001" "020130001001001" "040019426001001"
## [3,] "010010201001002" "020130001001002" "040019426001002"
## [4,] "010010201001003" "020130001001003" "040019426001003"
## [5,] "010010201001004" "020130001001004" "040019426001004"
## [6,] "010010201001005" "020130001001005" "040019426001005"
```

Improving Speed

The previous versions of CensusAPI2000 and CensusAPI2010 would, upon gathering all of the appropriate FIPS codes for the specified state, make one API call for every single unit of the specified geographic level. For instance, if the user was gathering data at the county level, and there were 88 counties in the given state, 88 API calls would be made. If there were 259,777 blocks in a given state, 259,777 API

calls would be made. Fortunately, the Census API provides a method for gathering data for every county or tract in a given state, just by providing the FIPS code of the state. The API can be used to extract data for every block group in a specific county, and every block within a specific tract. In terms of efficiency, this indicates that in order to extract data for the block level, there will need to be as many API calls as there are tracts in the state. Each API call gathers the data for every block in the specific tract, and after every tract in the state has been sent through the API call, every block in the state will be accounted for.

Overview of CensusAPI2010

General Structure

This version of CensusAPI2010 is capable of handling multiple states in a single function call. For instance, if the user wants to obtain the number of housing units in each tract for North and South Carolina, only one function call is required and a large data frame with all the appropriate data in both states is returned. The general structure of CensusAPI2010 function involves a wrapper function and two sub functions. The first sub-function, CensusData2010.sub, actually takes the user input and collects the appropriate data from the Census Bureau's API, and returns and formats it in a convenienent data frame. The second subfunction, wrapperCD2010, is built to handle the cases for which data in more than one state is requested. The function wrapperCD2010 calls the CensusData2010.sub function for each state requested, and puts all of the pieces for each state together into one convenient data frame and returning it to the user.

```
CensusAPI2010<-function(variables,state.fips,level=c("county","tract","
block group","block","cdp"),key,summaryfile=c("sf1","ACS"))
  {
```

The function header is defined, with all of the information necessary being required for the wrapper function and simply passed in. The object fips2010, a large matrix containing every FIPS code at every geographic level in the United States is loaded from R-Forge and GitHub, servers where code and data can be loaded into R. In the near future, it will be published on CRAN, the server/network where users can put code, data, and documentation for public use in R.

```
CensusData2010.sub<-function(variables,state.fips,level=c("county","tra
ct","block group","block","cdp"),key,summaryfile=c("sf1","ACS"))
  {
    bf<-function(){
    utils::data(fips2010,envir =parent.frame())
    assign("temp",fips2010)
    temp
    }
    fips2010<-bf()
```

Next in CensusData2010.sub, it is necessary to extract the appropriate FIPS codes for the selected geographic level. There two possible data sets to extract from, the ACS (American Community Survey) and sf1 (Summary File 1). The ACS contains more sensitive information about income, time traveled to work, the Gini Index, and educational attainment. Summary file 1 (the decennial Census long form) contains information pertaining to age, ethnicity, gender, family size, and type of housing. Likely for privacy reasons, ACS data is only available at the county level whereas Summary File 1 data is available at every geographic level. As a result, a check is put in place that will stop the function and print a notification if the user attempts to access ACS data at any geographic level except for county. Next, the function ensures that the user has entered a valid option for geographic level and the summary file.

```
if(level!="county"&&summaryfile=="ACS")
    stop("ACS data only available at county level. Change level to coun
ty to continue.")
    level<-match.arg(level,several.ok=FALSE)
    summaryfile<-match.arg(summaryfile,several.ok=FALSE)
    if(level!="cdp")
        fips<-fips2010[,which(colnames(fips2010)==state.fips)]
    if(level=="cdp")
    {
        fips<-unique(fips2010[which(substr(fips2010[,52],1,2)==state.fi
ps),"cdpfips"])
        fips.subset<-unique(substr(fips,3,7))
    }
    if(level=="county")
            fips<-unique(substr(fips,1,5))
    if(level=="tract")
            fips<-unique(substr(fips,1,11))
    if(level=="block group")
        {
            fips<-unique(substr(fips,1,12))
            fips<-fips[!is.na(fips)]
            #because we need to specify a county
            fips.subset<-unique(substr(fips,3,5))
        }
    if(level=="block")
        {
            fips<-unique(fips[!is.na(fips)])
            #because we need to specify a tract
            fips.subset<-unique(substr(fips,3,11))
        }

    #Clear out the NAs
    fips<-fips[!is.na(fips)]
```

The data containing all of the FIPS codes in the United States is organized in a matrix where each state is given one column. Within each column is the list of each FIPS code at the block level for that particular state. It is possible to deduce all of the FIPS codes at any geographic level from the block level codes by taking specific parts of each FIPS code and removing repeats. The appropriate column is selected, and based on the geographic level, specific digits within the full block FIPS are extracted. The block group and block levels save an object as fips.subset, which is used to aggregate smaller geographic levels of a state. Recall that the Census API does not

allow users to pull out data for every block or block group in a state. Thus, it is necessary to iterate through all of the tracts and counties in that state, respectively. The fips.subset object contains the FIPS codes at the tract or county level and is used for iteration. For example, at the block level, the fips.subset object will contain all of the tract level FIPS in the selected state. Iterating through all the tracts and combining the block level data in each tract will result in data being gathered for every block in the selected state.

An issue with data storage arose due to the fact that each state has a different number of blocks within it. The data had to be stored in such a way that the block level FIPS for any state could be accessed as a vector, given just the 2 digit FIPS code for that particular state. There are 52 vectors (50 states, Washington D.C., and CDP), each of different lengths. The appropriate amount of NAs were added to each vector except the longest one to make the length of every vector the same: the length of the longest one. Although somewhat inefficient, this allows for the matrix to be saved as a 3-megabyte matrix and conveniently accessed. This is why the commands to remove NAs from the vector of FIPS are necessary.

Next, the subfunction APIcall is defined. This is the method for actually extracting the data from the Census API. This version of the APIcall subfunction is different from the last version in that it only takes one argument: the FIPS for the desired geographic area. The selected demographic variables and geographic level are already input in the parent function and are able to be accessed. In order to extract data from the Census Bureau API, a URL containing details for the data to be

extracted is concatenated and includes the level at which it is to be extracted, from which database it is to be gathered, and the Census designated key for the particular user. For instructions on how to get your own Census Bureau key, see the end of this document. Based on the entered information, once the URL is constructed, the API can be accessed, and the appropriate data downloaded and formatted. When completed, the URL will look something like:

```
## [1] "http://api.census.gov/data/2010/sf1?get=P0010001&for=county:*&i
n=state:27&key=YOUR_KEY_HERE"
```

Given the URL above, the subfunction APIcall employs the rjson package in R to download the data provided by the Census Bureau in a JSON file. In the previous version of CensusAPI2010, the very rapid API calls caused by the inefficiencies would cause the Census Bureau to result in an error and CensusAPI2010 would crash. As an added feature of reliability, in the event that the API call returns as an error, the use of the "try-error" command will run the API call again. If another error returns, CensusAPI2010 will stop and print a warning advising the user to double check that the demographic variable entered is correct. As this was not included in the previous generation of CensusAPI2000 and CensusAPI2010, there is an additional element of user friendliness and convenience. Finally, if there are no errors, the JSON file is formatted and returned from the subfunction APIcall.

```
  APIcall<-function(fipscode)
  {
    if(level=="county"&&summaryfile=="sf1")
        url<-paste("http://api.census.gov/data/2010/",summaryfile,"?get
=",gsub(", ",",",toString(variables)),"&for=county:*&in=state:",state.f
ips,"&key=",key,sep="")

    else if(level=="county"&&summaryfile=="ACS")
```

```
        url<-paste("http://api.census.gov/data/2010/acs5?get=",gsub(",
","",toString(variables)),"&for=county:*&in=state:",state.fips,"&key="
,key,sep="")

    else if(level =="tract")
        url<-paste("http://api.census.gov/data/2010/",summaryfile,"?get
=",gsub(", ","",toString(variables)),"&for=tract:*&in=state:",state.fi
ps,"+county:",substr(fipscode,3,5),"&key=",key,sep="")

    else if(level =="block group")
        url<-paste("http://api.census.gov/data/2010/",summaryfile,"?get
=",gsub(", ","",toString(variables)),"&for=block+group:*&in=state:",st
ate.fips,"+county:",fipscode,"&key=",key,sep="")

    else if(level =="block")
        url<-paste("http://api.census.gov/data/2010/",summaryfile,"?get
=",gsub(", ","",toString(variables)),"&for=block:*&in=state:",state.fi
ps,"+county:",substr(fipscode,1,3),"+tract:",substr(fipscode,4,9),"&key
=",key,sep="")

    else if(level=="cdp")
        url<-paste("http://api.census.gov/data/2010/",summaryfile,"?get
=",gsub(", ","",toString(variables)),"&for=place:*&in=state:",state.fi
ps,"&key=",key,sep="")

    document <- try(rjson::fromJSON(file=url),silent=TRUE)
    if(class(document)=="try-error")
        {
            stop("Data could not be found.  Make sure to check http://a
pi.census.gov/data/2010/sf1/variables.html to see if you have the right
variable name")
        }
    m<-matrix(unlist(document),ncol=length(document[[1]]),byrow=TRUE)
    colnames(m)<-m[1,]
    m<-rbind(m[2:NROW(m),])
    m<-as.data.frame(m,stringsAsFactors=FALSE)
    return(m)
    }
```

Next, it is necessary to set up the empty data frame to 'fill in' with demographic data. This is for efficiency purposes; it is faster to fill in a blank template than to continually be integrating data together.

```
      d<-matrix(c(fips,rep(rep(NA,length(fips)),length(variables))),nro
w=length(fips))
    d<-as.data.frame(d,stringsAsFactors=F)
```

```
    colnames(d)<-c("fips",c(variables))
    rownames(d)<-fips
```

After the blank template has been created, it must be filled in. Depending on the geographic level entered, one of the options displayed below will be selected. Note that for the county and tract levels, only one call to the subfunction APIcall is needed as the data at these levels can be extracted for an entire state at once. The data frame returned from APIcall is saved as the object "m," the FIPS codes from the blank template and from the data frame "m" are matched together, and the data is put in the appropriate place. For the block group and block levels, the subset of data for the counties and tracts in fips.subset is consolidated into the blank template in the appropriate place. For convenience, there is a command to ensure the variables are reported as numeric instead of characters or factors. The data frame is then returned from CensusAPI2010.

```
  if(level=="county")
        {
            m<-APIcall(fipscode=state.fips)
            d[,2:(length(variables)+1)]<-m[match(paste(m$state,m$county
,sep=""),rownames(d)),1:length(variables)]
        }

    if(level=="tract")
        {
        m<-APIcall(fipscode=state.fips)
        d[,2:(length(variables)+1)]<-m[match(paste(m$state,m$county,m$t
ract,sep=""),rownames(d)),1:length(variables)]
        }

    if(level=="block group")
        {
            for(j in 1:length(fips.subset))
            {
            m<-APIcall(fipscode=fips.subset[j])
            d[substr(d$fips,3,5)==fips.subset[j],2:(length(variables)+1
)]<-m[match(paste(m$state,m$county,m$tract,m[,"block group"],sep=""),ro
wnames(d[substr(d$fips,3,5)==fips.subset[j],])),1:length(variables)]
```

```
            }
        }

    if(level=="block")
    {
        for(j in 1:length(fips.subset))
        {
            m<-APIcall(fipscode=fips.subset[j])
            d[substr(d$fips,3,11)==fips.subset[j],2:(length(variables)+
1)]<-m[match(paste(m$state,m$county,m$tract,m$block,sep=""),rownames(d[
substr(d$fips,3,11)==fips.subset[j],2:(length(variables)+1)])),1:length
(variables)]
        }
    }

    if(level=="cdp")
    {
        m<-APIcall(fipscode=state.fips)
        d[,2:(length(variables)+1)]<-m[match(paste(m$state,m$place,sep=
""),rownames(d)),1:length(variables)]
    }

    for(k in 2:(NCOL(d))){d[,k]<-as.numeric(d[,k])}

    return(d)
}
```

The last part of CensusAPI2010 is built to handle multiple states in one function call. This can be accomplished by assigning the argument state.fips as a character vector of state FIPS codes. The code below will call CensusData2010.sub for each state in the vector of state FIPS and consolidate the data from multiple states into one data frame. This allows for easier analysis of a specific region or group of states.

```
wrapperCD2010<-function(variables,state.fips,level=c("county","tract","
block group","block","cdp"),key,summaryfile=c("sf1","sf3"))
{

  if(length(state.fips)>1)
    {
        out<-lapply(state.fips,function(x){
    CensusData2010.sub(variables=variables,state.fips=x,level=level,key
=key,summaryfile=summaryfile)
```

```
        })
        fout<-data.frame(matrix(NA,ncol=NCOL(out[[1]]),nrow=sum(sapply(
out,NROW))))
        colnames(fout)<-colnames(out[[1]])
        rownames(fout)<-unlist(sapply(out,rownames))
        num<-sapply(out,NROW)
        num2<-c(1,num)
        num3<-cumsum(num2)

        for(i in 1:(length(num3)-1)){
        fout[seq(from=num3[i],to=num3[i+1]-1),]<-as.matrix(out[[i]])
        }
        for(k in 2:(NCOL(fout))){fout[,k]<-as.numeric(fout[,k])}
        return(fout)
    }

CensusData2010.sub(variables=variables,state.fips=state.fips,level=leve
l,key=key,summaryfile=summaryfile)
}
```

Examples of Use & R Packages

CensusAPI2010 is available in the 'UScensus2010' R package currently maintained by Professor Zack Almquist on R-Forge and GitHub, along with all of the FIPS data necessary. It is intended for the functions and FIPS codes discussed in this report to be published on CRAN. In addition, there is a CensusAPI2000 function that works for the 2000 decennial census in a very similar way to the 2010 version. Using these functions in conjunction allows for comparison of demographic information over time and allows for important analysis.

Examples of Use

```
mn.population.county<-CensusAPI2010(variables=c("P0010001","P0120002","
P0120026"),state.fips="27",level="county",key=key,summaryfile="sf1")
head(mn.population.county)
```

```
##        fips P0010001 P0120002 P0120026
## 27001 27001    16202     8205     7997
## 27003 27003   330844   165327   165517
## 27005 27005    32504    16314    16190
```

```
## 27007 27007    44442    22184    22258
## 27009 27009    38451    19277    19174
## 27011 27011     5269     2575     2694
```

```
mn.population.tract<-CensusAPI2010(variables=c("P0010001","P0120002","P
0120026"),state.fips="27",level="tract",key=key,summaryfile="sf1")
head(mn.population.tract)
```

```
##                     fips P0010001 P0120002 P0120026
## 27001770100 27001770100     2327     1211     1116
## 27001770200 27001770200     2336     1189     1147
## 27001770300 27001770300     3268     1548     1720
## 27001770400 27001770400     3052     1566     1486
## 27001790501 27001790501     1957     1027      930
## 27001790502 27001790502     3262     1664     1598
```

```
mn.acs<-CensusAPI2010(variables="B06011_001E",state.fips="27",level="co
unty",key=key,summaryfile="ACS")
head(mn.acs)
```

```
##        fips B06011_001E
## 27001 27001       21570
## 27003 27003       33910
## 27005 27005       24060
## 27007 27007       20729
## 27009 27009       26361
## 27011 27011       21393
```

Get Your Own Census Bureau Key

In order to use CensusAPI2000 and CensusAPI2010, you will need your own Census Bureau issued key. To obtain one, go to 'http://api.census.gov/data/key_signup.html' and enter your organization and email address. Read the terms of service, and if you agree, click 'agree' and the Census Bureau will send you an email. Click the link in the email to obtain and activate your key. It is useful to save your key as an object in R called 'key' and save it as an easy to access .rda file.